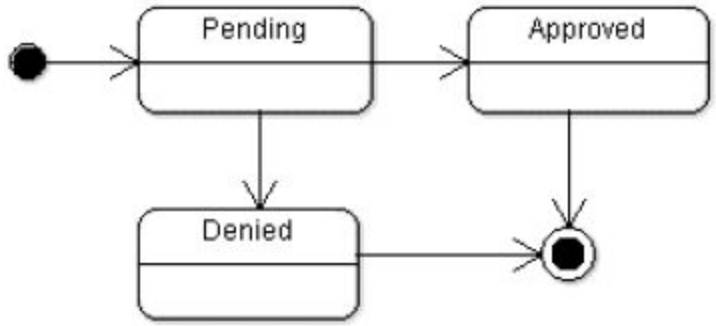
Project 1: Expense Reimbursement System

Executive Summary

The Expense Reimbursement System (ERS) will manage the process of reimbursing employees for expenses incurred while on company time. All employees in the company can login and submit requests for reimbursement and view their past tickets and pending requests. Finance managers can log in and view all reimbursement requests and past history for all employees in the company. Finance managers are authorized to approve and deny requests for expense reimbursement.

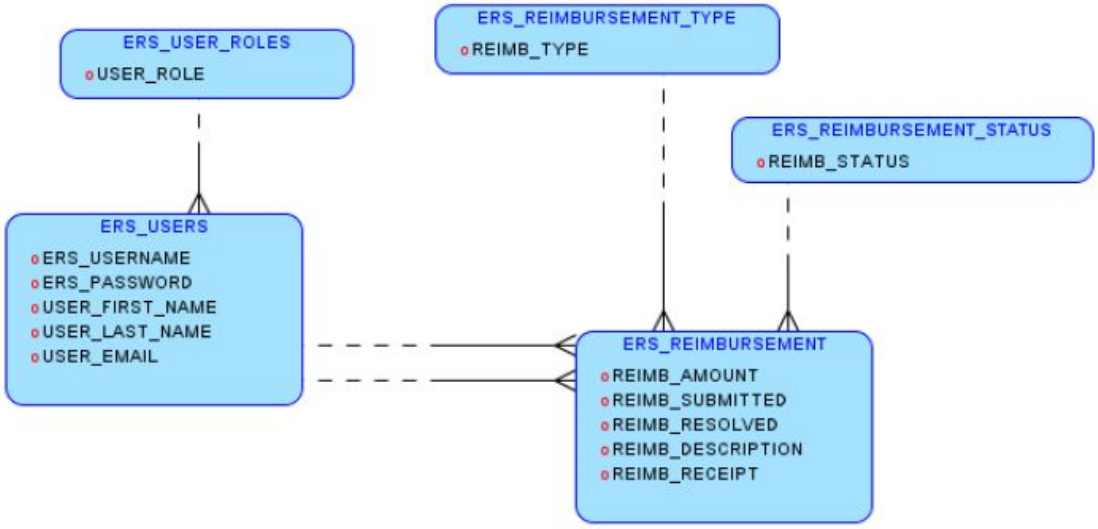
State-chart Diagram (Reimbursement Statuses)



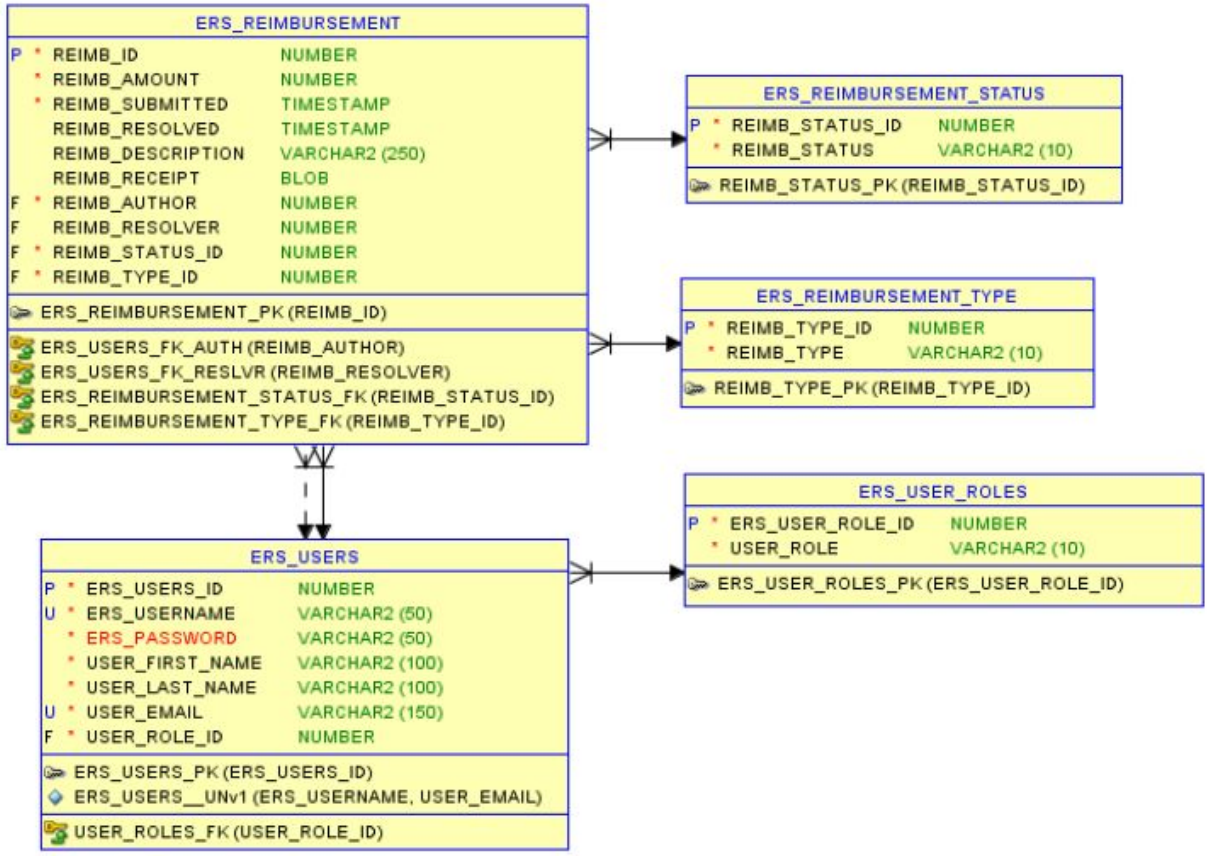
Reimbursement Types

Employees must select the type of reimbursement as: LODGING, TRAVEL, FOOD, or OTHER.

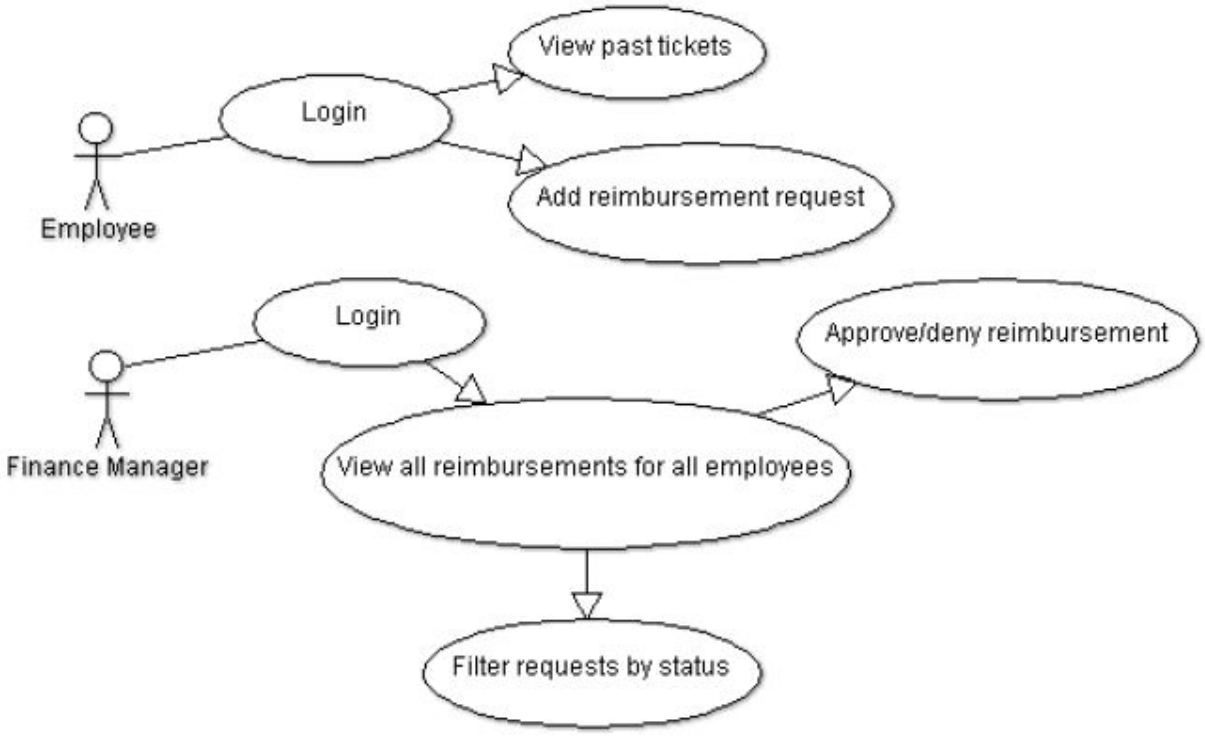
Logical Model



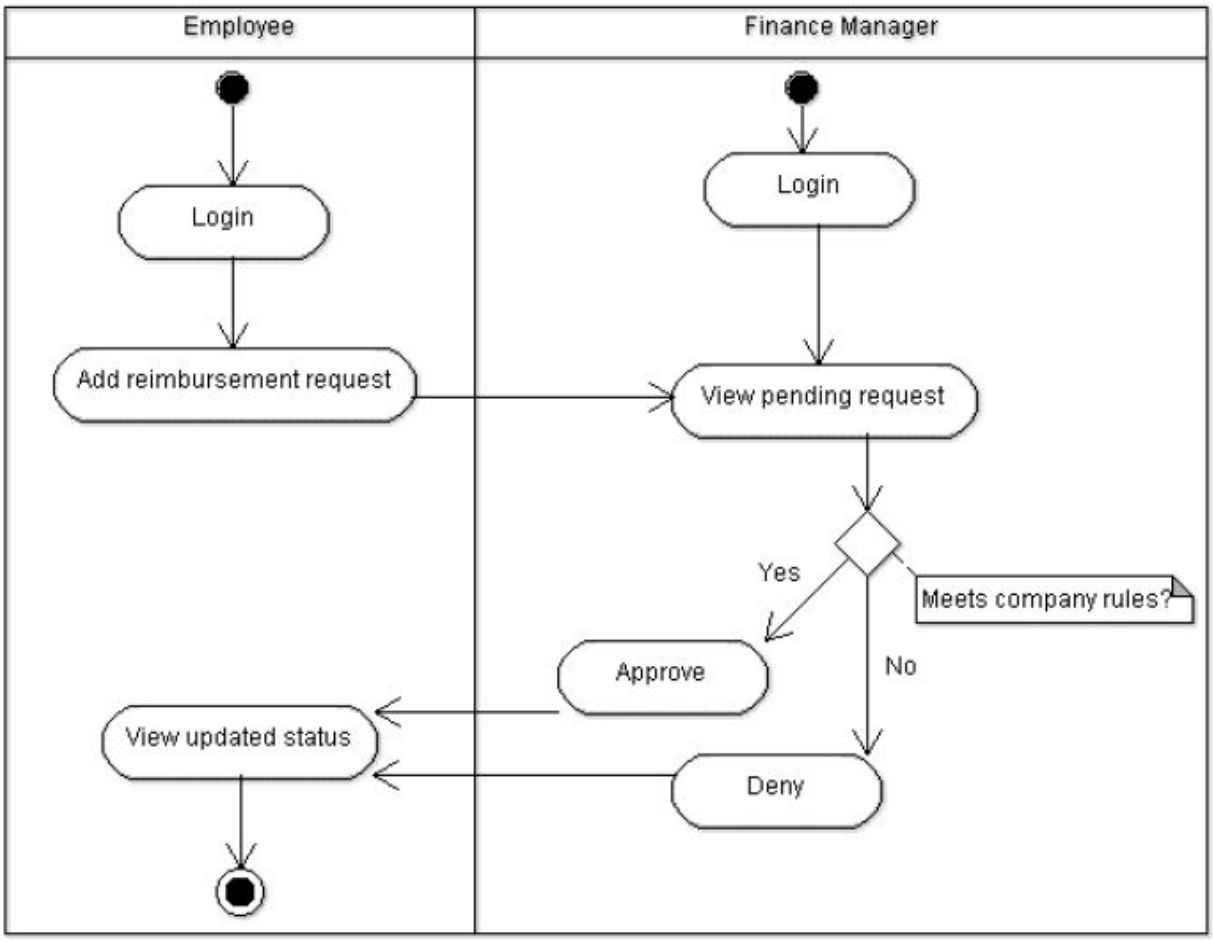
Physical Model



Use Case Diagram



Activity Diagram



Technical Requirements

- The application shall employ the DAO design pattern, and properly separate your code into the appropriate layers
- The back-end system shall use JDBC to connect to a Postgresql database.
- The application shall deploy onto a Tomcat Server
- The middle tier shall use Servlet technology for dynamic Web application development
- The front-end view can use JavaScript and use AJAX to call server-side components. The web pages should look presentable (try using css and bootstrap); I'd rather not see a website from 1995.
- Use Log4J and JUnit.

- Passwords should be encrypted in Java and securely stored in the database
- Users can upload a document or image of their receipt when submitting reimbursements(optional)
- The application will send an email to employees letting them know that they have been registered as a new user, giving them their temporary password(optional)

**Please take the deadline seriously.

**Do NOT spend too much time stuck on a single blocker without asking a batch-mate for help.

Tips on how to start

If you're having trouble wrapping your head around how to start, here is my preference for starting project 1:

1. Start with your SQL schema. Create all your tables (entities), entity relationships, sequences, triggers, functions, and stored procedures. You should be able to populate your entities with data before even touching java or html; attempt to run simple CRUD operations on each of your tables to verify that your triggers are firing.
    a. The "User Role", "Reimbursement Type", and "Reimbursement Status" tables are all LOOK UP TABLES (enum values). ATTENTION: the Database Administrator (you) will need to pre populate these look up tables with data before doing anything else; because they will have not null foreign keys pointing them.
    b. These look up tables will likely not have DAOs when you get to java code.
2. Once your database is working properly, go to java and create your model layer. So create the necessary object representations of your database tables.
    a. In our case we'll be creating objects for "User", "Reimbursement", "ReimbursementStatus", "ReimbursementType", and "UserRole"
3. Once your models exist, set up your DAOs using JDBC. Don't make a service layer...don't make any controllers or servlets or html or anything else….JUST your DAOs. Make sure your DAO layer's basic crud methods work.

    a. Just write a quick main method that calls each of the DAO methods separately then go to SQL Developer to make sure they worked.
    b. Note: you will likely need more DAO methods later, as your develop you'll realize that more specific queries are necessary in certain instances. BUT step 2 is about creating the basics for now.
4. Onces my DB schema and DAOs are functioning, I like to create my service layer while using Test Driven Development (TDD).
    a. Create simple service methods for now, until you get further into development and realize what specific methods you'll need. Create simple methods like "verifyLoginCredentials(String uname, String pass)", "retrieveReimbursements(User user)", "registerUser(User user)", etc.
5. Once my service layer is up and running, I like to create my servlets using a front controller design pattern; again, employing TDD strategies.

Once I'm done with servlets then my server is ready; now I can start on the front end. Again, this is a guideline for starting the project. This is not the only way to start the project but if you don't know how to start I'd say use this guideline.